Jacobian 神经网络算法

甄景贤 (King-Yin Yan)

General.Intelligence@Gmail.com

经典的神经网络 Back Prop 学习算法，它是一个 error-driven 算法，但在很多人工智能的实际应用中，不存在唯一的「理想答案」，而是根据正或负的奖励 (reward) 学习。当答案正确时，奖励 $> 0, \mathrm{error} = 0$；当答案不正确时，奖励 < 0，但 error 仍是不知道的（因为不知道理想答案）。简言之，就是不能用 error-driven 学习。

所以我想出了一个 reward-driven 的学习法：假设神经网络将 $\boldsymbol{x}_0 \mapsto \boldsymbol{y}_0$，它通常也会将 $\boldsymbol{x}_0$ 的邻域 map 到 $\boldsymbol{y}_0$ 的邻域。如果我们想「加强」这个映射，可以将「更大的 $\boldsymbol{x}_0$ 的邻域」映射到「接近 $\boldsymbol{y}_0$ 的邻域」。

这种算法对人工智能应该很重要，暂时我还想不出有什么其他办法，可以做到 [深度] 神经网络的 reward-driven 学习。

将这思想更准确化，可以将 feed-forward 神经网络的构造看成是这样的：

$$\boldsymbol{y} = \boldsymbol{F}(\boldsymbol{x}) \tag{1}$$

$$\boldsymbol{y} = \sigma \overset{L}{W} \ldots \sigma \overset{\ell}{W} \ldots \sigma \overset{1}{W} \boldsymbol{x} \tag{2}$$

其中 W 代表每一层 (layer) ℓ 的矩阵。

（下面会用到）F 的**反方向**是：

$$\boldsymbol{x} = \boldsymbol{F}^{-1}(\boldsymbol{y}) \tag{3}$$

$$\boldsymbol{x} = \overset{1}{\rotatebox{90}{W}} \reflectbox{σ} \ldots \overset{\ell}{\rotatebox{90}{W}} \reflectbox{σ} \ldots \overset{L}{\rotatebox{90}{W}} \reflectbox{σ} \, \boldsymbol{y} \tag{4}$$

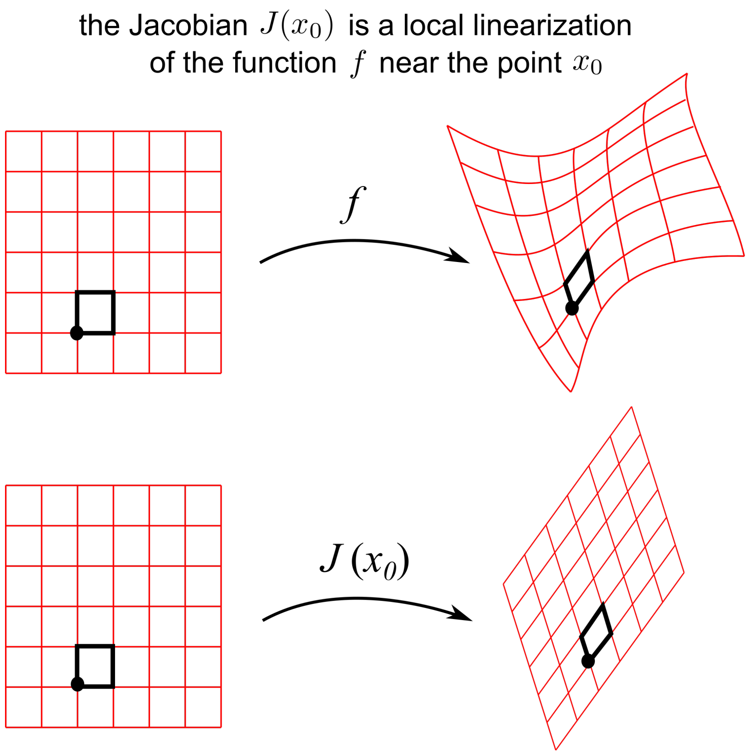
注意：$\rotatebox{90}{$W$} = W^{-1}$, $\reflectbox{$\sigma$} = \sigma^{-1}$，形状不同。

假设在 $\boldsymbol{x}$ 空间有体积元 U，经过 $\boldsymbol{F}$ 变换成 $\boldsymbol{y}$ 空间的体积元 V，那么：

$$U = |J| \cdot V \tag{5}$$

$J = \left[\dfrac{\partial \boldsymbol{F}(x)}{\partial x} \right]$ 叫 Jacobian 矩阵。

在我们的情况下，$|J| = \left|\frac{\partial \boldsymbol{F}^{-1}(\boldsymbol{y})}{\partial \boldsymbol{y}}\right|$ 在 $\boldsymbol{y}_0$ 的值，代表「单位体积元由 $\boldsymbol{y}_0 \mapsto \boldsymbol{x}_0$ 的变化率」。（下面会看到，$\boldsymbol{F}$ 和 $\boldsymbol{F}^{-1}$ 的正/反方向不太重要，因为基本上不影响计算复杂度。）



$$(6)$$

每次得到**正奖励**，我们会令 Jacobian $|J|$ 增加一点：

$$|J| := \det \left[\frac{\partial \boldsymbol{F}^{-1}(\boldsymbol{y})}{\partial \boldsymbol{y}}\right]_{n\times n} \tag{7}$$

下标表示那是一个 $n \times n$ 矩阵。

其实 Jacobian 矩阵的意义就是：

$$J = \left[\frac{\partial \text{ 输出}}{\partial \text{ 输入}}\right] \tag{8}$$

神经网络的输入和输出都是 $\dim n$，所以 Jacobian 很自然是 $n \times n$ 矩阵。

用**梯度下降法**，我们需要计算这些梯度：$\left[\frac{\partial |J|}{\partial \boldsymbol{W}}\right]$，总数是网络中的 weights 的个数 $= \sum m_\ell$。

要用到 determinant 的微分公式：

$$\frac{d}{dt}|A(t)| = tr(\mathrm{adj}(A) \cdot \frac{dA(t)}{dt}) \tag{9}$$

$$\mathrm{adj}(A) := |A| \cdot A^{-1} \tag{10}$$

换句话说，对於每个权重 $w := \overset{\ell}{W}_{ij}$，我们要计算：

$$\frac{\partial}{\partial w}|J| = tr(|J| \cdot J^{-1} \cdot \left[\frac{\partial J}{\partial w}\right]) \tag{11}$$

注意：$|J|$ 和 J^{-1} 是 $\boldsymbol{y}_0$ 的函数，只需在大 loop 外一次过计算。

问题是，计算 $\left[\dfrac{\partial J}{\partial w}\right]_{n\times n}$ 的时候：

$$\frac{\partial J}{\partial w} = \frac{\partial}{\partial w}\frac{\partial \boldsymbol{F}^{-1}}{\partial y} = \frac{\partial}{\partial w}\frac{\partial}{\partial \boldsymbol{y}} \overset{1}{\gtrsim} \oslash \ldots \overset{\ell}{\gtrsim} \oslash \ldots \overset{L}{\gtrsim} \oslash\, \boldsymbol{y} \tag{12}$$

这牵涉到用 w 对 W^{-1} 的分量微分，可以想像就算计了出来也会是极复杂的。解决办法是，索性「**本末倒置**」，用 $\gtrsim$ 来定义神经网络，然后在 forward propagation 时才用 $W = \gtrsim^{-1}$ 计算。

J 的分量写出来是：

$$J_{ij} = \frac{\partial \boldsymbol{F}_i^{-1}}{\partial y_j} = \frac{\partial}{\partial y_j}\left[\overset{1}{\gtrsim} \oslash \ldots \overset{\ell}{\gtrsim} \oslash \ldots \overset{L}{\gtrsim} \oslash\, \boldsymbol{y}\right]_i =: \nabla^1_{ij} \tag{13}$$

$$\begin{cases} \nabla^1_{ij} & := \displaystyle\sum_{k_1} [\, \overset{1}{\gtrsim}_{ik_1} \oslash'(y^2_{k_1}) \nabla^2_{ij}\,] \\ \nabla^\ell_{ij} & := \displaystyle\sum_{k_\ell} [\, \overset{\ell}{\gtrsim}_{k_{\ell-1}k_\ell} \oslash'(y^{\ell+1}_{k_\ell}) \nabla^{\ell+1}_{ij}\,] \\ \nabla^L_{ij} & := \qquad \overset{L}{\gtrsim}_{k_{L-1}j} \oslash'(y_j) \end{cases} \tag{14}$$

这情况完全类似於经典 Back Prop，以上只是 chain rule 的应用，∇^ℓ 将每层用 chain rule 分拆开来，所以 ∇ 又叫 “local gradient”。上式就是整个网络的**反向传递**，其中每个 weight 出现 exactly 一次。

但工作还未完，我们要计算 $\dfrac{\partial J_{ij}}{\partial \succcurlyeq} = \dot\nabla^1_{ij}$。（定义 $\succcurlyeq := \overset{\ell}{\gtrsim}_{gh}$, $k_0 := i$, $k_L := j$）

注意：$\boldsymbol{x} = \boldsymbol{F}^{-1}(\boldsymbol{y})$，所以 $\boldsymbol{y}$ 是**自变量**，$\succcurlyeq$ 不影响 $\boldsymbol{y}$，所以 $\dfrac{\partial \boldsymbol{y}}{\partial \succcurlyeq} \equiv 0$。

上面这句好像有问题：搞不清是 who depends on whom，以下可能错误。

$\succcurlyeq$ 必会是 $\gtrsim$ 的其中一元，但如果 $\succcurlyeq \notin \gtrsim$，以下的项微分后都会变成 0：

$$\begin{cases} \dot\nabla^1_{ij} = \displaystyle\sum_{k_1} \left[\, \overset{1}{\gtrsim}_{ik_1} \oslash'(y^2_{k_1}) \dot\nabla^2_{ij}\,\right] \\ \dot\nabla^\ell_{ij} = \displaystyle\sum_{k_\ell} \left[\, \overset{\ell}{\gtrsim}_{k_{\ell-1}k_\ell} \oslash'(y^{\ell+1}_{k_\ell}) \dot\nabla^{\ell+1}_{ij}\,\right] \\ \dot\nabla^L_{ij} = \qquad \overset{L}{\gtrsim}_{k_{L-1}j} \oslash'(y_j) \equiv 0 \end{cases} \tag{15}$$

所以实际上只剩下一项：

$$\frac{\partial J_{ij}}{\partial \succcurlyeq} = \sum_{k_1} \left[\overset{1}{\gtrsim}_{k_0k_1} \oslash'(y^2_{k_1}) \ldots \sum_{k_{\ell-2}} \left[\overset{\ell-2}{\gtrsim}_{k_{\ell-3}k_{\ell-2}} \oslash'(y^{\ell-1}_{k_{\ell-2}}) \ldots \right.\right. \tag{16}$$

$$\begin{cases} \ldots \overset{\ell-1}{\gtrsim}_{k_{\ell-2},g} \oslash'(y^\ell_g) \oslash'(y^{\ell+1}_h) \nabla^{\ell+1}_{ij} \,]] \\ \ldots \overset{\ell-1}{\gtrsim}_{k_{\ell-2},g} \oslash'(y^\ell_g) \oslash'(y^{\ell+1}_h) \,]] & \text{if } \succcurlyeq \in \text{last layer} \end{cases}$$

上式的意思是：每层 layer 重复一块 $[\sum \gtrsim \oslash']$，直到遇到 $\succcurlyeq = \overset{\ell}{\gtrsim}_{gh}$，则用结尾形式取代之。

和经典 Back Prop 不同的是，上式只是 $n \times n$ 矩阵中的一个元素，从复杂度而论，每个 weight 的 ∇ 计算，增加了起码 n^2 倍的复杂度（虽然其计算上可以共用一些结果）。记住 $n = \dim \boxed{\text{状态空间}}$。

可以这样理解：每个 weight 的调教，需要计算这个 weight 对 Jacobian 的影响，而那 Jacobian 是**整个**网络的特性。关键似乎就在於每个 weight 对 Jacobian 的**影响**。

现在回看更高层次的这个式子：

$$\frac{\partial}{\partial w}|J| = tr(|J| \cdot J^{-1} \cdot \left[\frac{\partial J}{\partial w}\right]) \tag{17}$$

$$= |J| \cdot tr(\left[\frac{\partial y}{\partial x}\right] \cdot \left[\frac{\partial}{\partial w}\frac{\partial x}{\partial y}\right]) \tag{18}$$

$$= |J| \cdot \sum_{ij} \left(\frac{\partial y}{\partial x}\right)_{ij} \left(\frac{\partial}{\partial w}\frac{\partial x}{\partial y}\right)_{ij} \tag{19}$$

上式中最重要（最慢）的是那 $(i,j) \in n \times n$ 求和。裡面的第一個因子是 Jacobian J，第二个因子是我们刚计算了的 $\nabla_w J^{-1}$。

Back Prop 的 ∇ 形式上是 $\dfrac{\partial\ \text{输出}}{\partial\,\mathrm{w}}$，我们的 ∇ 形式是 $\left[\dfrac{\partial}{\partial w}\dfrac{\partial\ \text{输入}}{\partial\ \text{输出}}\right]_{n\times n}$。

其实我们只需要计算 $\nabla_w|J|$ 的**大约方向**。暂时我在代码中的做法是：忽略式 (19) 中较小的项，那就不需做足 n^2 个乘积。

或者可不可以将 $|J|(\mathbb{w})$ 看成是一个 weight $\mathbb{w}$ 的函数，然后用它的 Taylor series expansion 来近似？

其实更大的问题是：给定一个输入 x，整个 F 只能输出一个值，因而不能做到 stochastic policy。 暂时搁置。

Notes

– Parameters are organized hierarchically (“deep”)

– Jacobian of F at x in terms of W is easy to calculate

–

References